Can Hot Traces Improve Value Prediction?

Assad Khamaisee[(1)], Avinoam Kolodny[(1)] and Avi Mendelson[(2)]

[(1)] EE department, Technion, [(2)] Intel Corporation, Israel

Abstract

This paper focuses on the interrelations between two well-known techniques for processor performance acceleration: trace cache (in particular "hot traces") and value prediction. Value prediction was proven to have potential for performance improvement, but it suffers from the need to use relatively large tables and it needs an accurate classification mechanism to indicate which instructions are most likely to be predicted correctly in order to minimize the amount of mispredictions. Trace cache was proposed as a mechanism to improve the effective fetch bandwidth. Recently, several papers proposed to use "hot traces" (traces that have been used repeatedly), as a basis for different trace cache optimizations such as reducing the power consumption of the core and gaining better performance given a limited-size cache.
This paper proposes to use hot traces as a filter to the value prediction, so that the value prediction will be done only for instructions belonging to hot traces. We will show that the new proposed technique improves the prediction accuracy for most of the applications, and reduces the miss ratio of value prediction.

1. Introduction

In order to extract more performance out of single threaded applications, various techniques were proposed, including the use of trace cache and value prediction. Trace cache was introduced as a mechanism for increasing the fetch bandwidth of out-of-order machines [rot96], while Value prediction was proposed for overcoming the memory wall problem [lip96a] (by predicting LOAD instructions), or for increasing the effective parallelism of the machine [lip96b,gab98] (by predicting arithmetic operations). Different investigations showed [gab98] that the benefit of using value prediction is heavily dependent on the size of prediction tables and the quality of the filtering being used.

Different methods have been proposed for value prediction. In this work we will refer to three of them: last value, stride predictor and 2-level value predictor. Last value predictor, proposed in [lip96b], predicts that the value produced by an instruction will be the same as the value it produced last time. The implementation of the last value predictor includes two tables, one (VPT) that keeps the last produced value and another used as a classification table (CT) to decide whether the predicted value should be considered or not.

The stride predictor, presented in [Gab98], uses a similar structure as the last value predictor, but instead of predicting based on last value, it tries to identify a pattern of stride (can be 0) and to predict the next outcome based of the stride values. The stride method also keeps a VPT table that keeps the last value and the stride value, and a classification table. One can replace the classification table with a decision that only strides which are in some range are acceptable and all the rest are ignored.

The 2-level value predictor was presented in [wan97]. Here, the method is based on the observation that a substantial percentage of the dynamic instructions have 4 or fewer unique values in their recent history, by storing their appearance count the predictor chooses 1-out-of-4 predictions based on 2 levels prediction technique similar to the method used for

branch prediction. Note that each entry of the table needs to keep 4 different values, and so is much larger (in bytes) than the size of the entry in last value or in stride predictor.

In order to improve the predictability of value prediction and in order to reduce the size of the tables it uses, different filtering mechanisms were proposed: [cald99] looked at techniques to reduce the size of the different tables with minimal performance loss, while [tune01] [fiel02] suggest different heuristics of finding the critical path of the program and selectively predicting the values of instructions on the predicted critical path. These techniques require extra hardware that needs to be added to the processor for getting information from various subsystems which are timing sensitive. This paper examines a different approach: we would like to take advantage of trace cache techniques, which exist in high performance processors such as Pentium®-4.

Trace cache is a mechanism proposed for increasing the fetch bandwidth of the processor by storing and retrieving blocks of dynamically-adjacent instructions, which may contain one or more taken branches. These sequences of instructions are stored in contiguous cache-memory locations. In this paper we are using a trace cache which is similar to the one described in [rot96]; i.e., the trace cache mechanism has two stages: (1) if a trace is not found in the trace cache we gather instructions until it reaches some termination conditions and then stores it in the trace cache (2) if a trace is found in the trace cache we use it and while executing, branch conditions are checked. In case of "departure" from the predicted trace, we terminate the execution of the current trace and look for a new trace in the trace cache (or build a new one).

Recently, it was suggested to use a subset of all traces in a program, termed "*hot traces*", as a vehicle for various optimizations. A "*hot trace*" is defined as a trace which is used more than some threshold number of times during its life time in the cache, and [Kos02] shows that a relatively small number of hot traces are responsible for the majority of the instructions being executed by the processor. In [Ros01] hot traces are used as an optimization parameter for reducing the power of high performance computers and in [Ros03] it was suggested to use hot traces as "building block" for further optimizations in order to improve the performance of the machine.

This paper proposes to use hot traces as a filter to indicate for which instructions value-prediction should be applied. The motivation here is two-fold: in many cases hot traces represent the "regular" portion of the execution; i.e., inner loops, which may have higher probability for correct prediction of the values associated with instructions. Furthermore, the number of instructions predicted within the hot traces is relatively small, but yet a vast portion of the execution originates from them. Thus, value caches of modest size may be sufficient to serve a lot of the required predictions.

The rest of the paper will be organized as follows: Section 2 describes the characterization criteria we will use later on in the our comparison, Section 3 gives a brief description of our simulation environment, Section 4 presents the main results and we conclude with Section 5.

2. The Characterization Criteria

In order to characterize the new proposed technique, we look at the following parameters:

- ***N***: The total instructions count in the program.
- ***P***: The total predictable instruction count in the program. This set of instructions includes all the instructions which will be forwarded to the value prediction mechanism. In the "traditional" techniques, it will include all the instructions that "pass" the type criteria; e.g., load instructions if we predict only loads (In this work LOAD and Arithmatic Instruction are the type criteria). In the new proposed technique it will include all instructions which belong to hot traces and pass the type criteria.

- ***V***: (valid) Instructions that pass the CT filter; i.e., the predictor decides to "take a bet" and predict their data values. Note that ***V*** is always a subset of ***P***.
- ***S***: (success) Instructions that were predicted correctly. Note that ***S*** is always a subset of ***V***.

In order to characterize the different mechanisms, we will use the following ratios:

- ***PR***: The total predictable instructions rate (***P/N***).
- ***SQ*****:** Success quality (S/P) – how many predictions were found to be correct out of the total candidate.
- ***MR***: The total value prediction miss rate (***(V-S)/N***).

Note that when using the new proposed technique; i.e., using the hot traces to gate access to the value predictor, the above stated instruction groups (***P,V***) will be numerically smaller (for the same ***N***) due to the filtering, but we hope that the successfully predicted values (***SQ***) will increase and the overall miss ratio will be reduced.

3. The Simulation Framework

The simulation model environment is based on sim_outorder version of the simple scalar 3.0 simulation tool set, using PISA instruction set. The simulator was enhanced to include trace cache management, trace selection and hot trace identification algorithms. We also implement different value prediction mechanisms, three of them are presented in this paper.

In order to evaluate the different techniques, we use ten integer and floating point applications out of the CPU SPEC200 performance suite: *gzip, vpr, gcc, twolf, art,equake, ammp, mesa , perl and mcf.*

The simulated trace cache model consists of 8k trace entries, each trace includes up to 16 instructions and up to 4 basic blocks (maximum 4 branch instructions may be included in the same trace). The LRU (last recently used) replacement policy is used for cache space conflict. The simulated *trace utilization threshold* (***tuthr***); i.e., how many times we use the trace before declaring it as a hot trace, ranges from 2 to 100.

4. Experimental Results

The new proposed technique suggests to use the hot trace indicator as a filter for deciding "which instructions are important for the program"; i.e., execution paths which are executed repeatedly many times. Since the question of building efficient traces is out of the scope of this paper, and we are using the basic trace construction mechanism as was described in [rot96], we distinguish between the group of programs for which the current tracing mechanism works efficiently, and the rest of the benchmarks. In order to distinguish between these groups, we examine the predictability ratio of the program and examine how it changes with the value of the *trace utilization threshold (**tuthr***) that indicates after how many uses of a trace, we treat it as a hot trace. For each of these groups we examine different characteristics of the proposed mechanism.

4.1. PR Characterization

The ***PR*** (total predictable instructions ***P/N***) characterization indicates how many instructions, out of all instruction being executed are candidates for value prediction. When no hot trace selection is available, ***P*** is determined by the type of the instructions(Load and Arithmetic instructions). When hot trace selection is used, ***P*** is determined by the type of the instructions within hot traces only.

Figure 1 shows how the ***PR*** characterization that indicates how many instructions will be used as candidates for value prediction changes in respect to the threshold (***tuthr)*** of the "hot trace indicator".

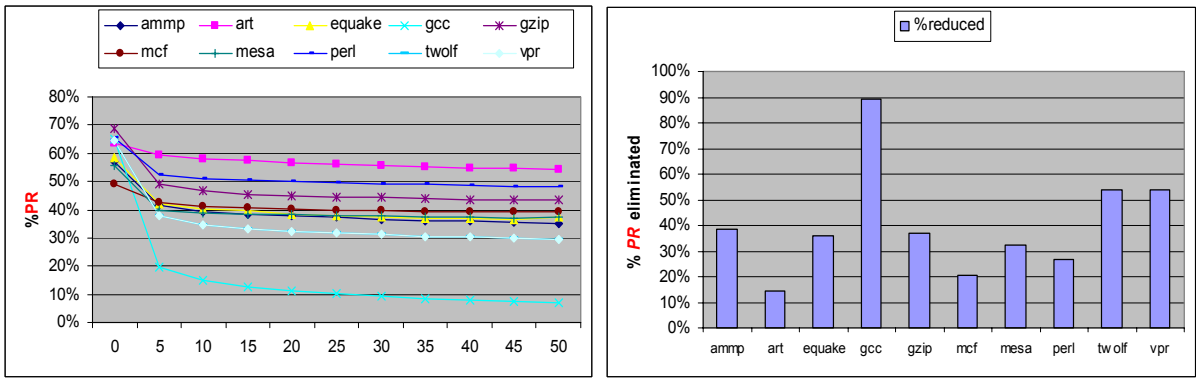

Figure 1. The ***PR*** characterization in respect to ***tuthr***

We can observe that the hot trace indicator reduces the ***PR*** for all programs. But for many of them, (1) the percentage of the instructions being eliminated is relatively small and (2) the slopeof the change after ***tuthr***=10 is very flat. The first effect could be either because the programs use a very small number of traces all the time, or because of a very inefficient way of building the traces, so hot traces have only modest impact. Since the analysis of the hot trace build is out of the scope of this paper, we decided to focus the discussion on those applications where the current trace build procedure seems to be effective; i.e., GCC, Twolf and VPR. We will examine the behavior of the rest of the applications separately.

4.2. Success Quality (SQ) – S/P

The success quality indicates how many correct predictions the value predictor made out of all the accesses to the value predictor mechanism. Let's start looking at the characterization of the ***SQ*** qualifier. Figure 2 shows how the ***SQ*** changes for the three selected programs in respect to the ***tuthr*** threshold, when a table with 1K entries is used. Please note that the left-most bar in each group is for ***tuthr***=0; i.e., no hot trace filter was used.

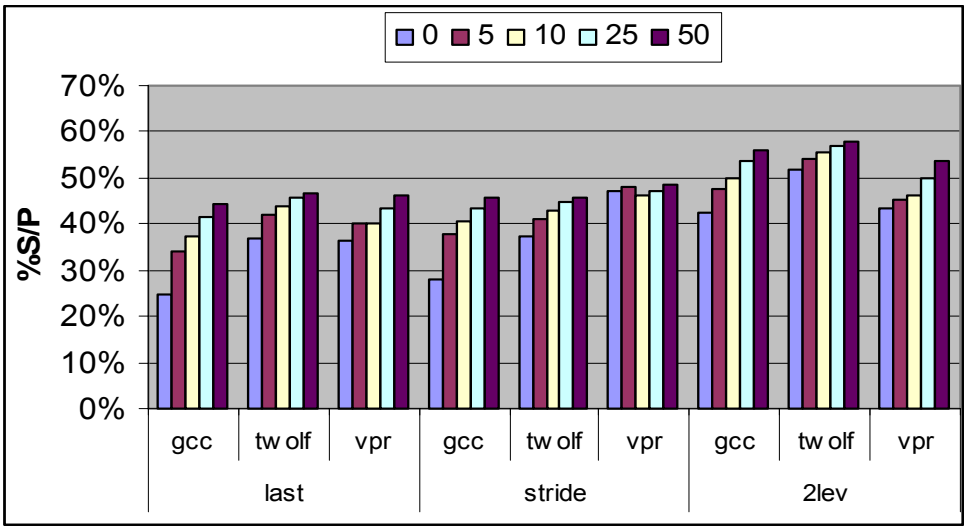

Figure 2: ***SQ*** characterization of well-filtered applications.

Figure 2 shows that for most of the cases, the ***SQ*** improves when we increase the ***tuthr*** parameter. It indicates that using the hot trace indicator improves the quality of the

prediction we are making. (but we also need to remember that it reduces the overall number of instructions it attempts to predict, so the total number of "correctly predicted" instructions can be reduced, but the confidence we have that an instruction we try to predict will be predicted correctly, increases). Figure 3 provides the same characterization for the rest of the program list we use.

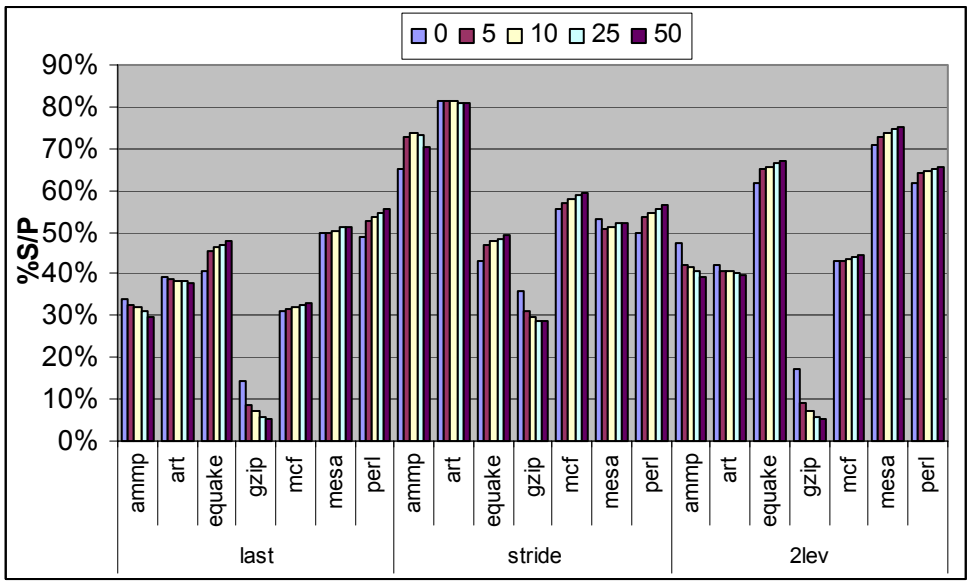

Figure 3: SQ characterization for other programs

We can observe that the trends and the observations are not coherent here. Two programs; ammp and gzip are getting less favorable prediction when using hot trace filter, ART and MCF are having relatively good predictions, but they are not affected by the hot trace selection. One may assume that the number of different traces in these program is small enough so any filter can only reduce the accuracy rather than improve it. Other programs show similar behavior as in figure 2.

4.3. Miss Ratio (MR)

A very important factor for a good value predictor mechanism is the amount of miss-prediction it causes. Please note that the number of misses is not equal to ***P-S*** (the amount of accesses to the predictor minus the number of success predictions), since many of the access to the value predictor will not be considered if they will be "rejected" by the CT mechanism. In this section we will focus on the ***MR*** characterization; i.e., the number of misses out of the total instructions the program executed. As before, we will separate the discussion to the "well filtered" applications and the rest.

Figure 4 indicates that the new proposed technique is very efficient for reducing the performance penalty caused by value miss predictions. The leftmost bar in each group indicates the miss prediction when no hot trace filter is used, the rest of the bars show the miss prediction for different values of the hot-trace threshold (***tuthr***). We can observe that as we increase the threshold, the miss ratio (***MR***) decreases. This trend is important in particular for 2lev predictor that suffers from very high level of misses, but it is also important for last and stride predictors where the miss-prediction is reduced from about 2% to about 0.5%.

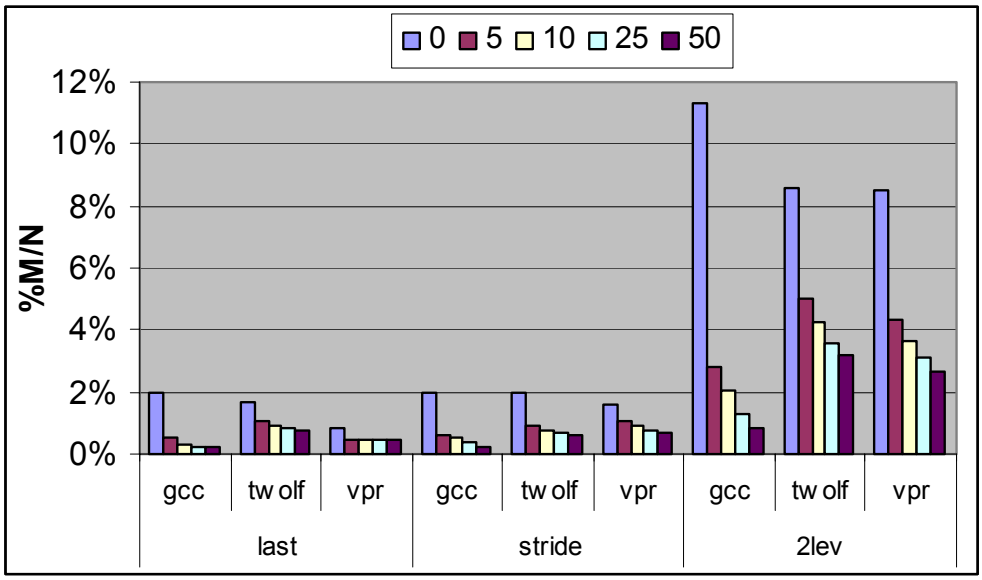

Figure 4: MR for well-filtered applications

Figure 5 presents the same view for the rest of the applications. As before, the picture here is more complicated. For most of the applications, and in particular for the 2lev implementation, the miss ratio improves significantly as we increase the ***tuthr*** parameter. Some applications such as perl and mcf, although their miss ratio is quite small in the first place, do not reduce it any more as we increase the threshold level. For some other applications, such as mesa for stride and last value the improvement is less significant than we could expect.

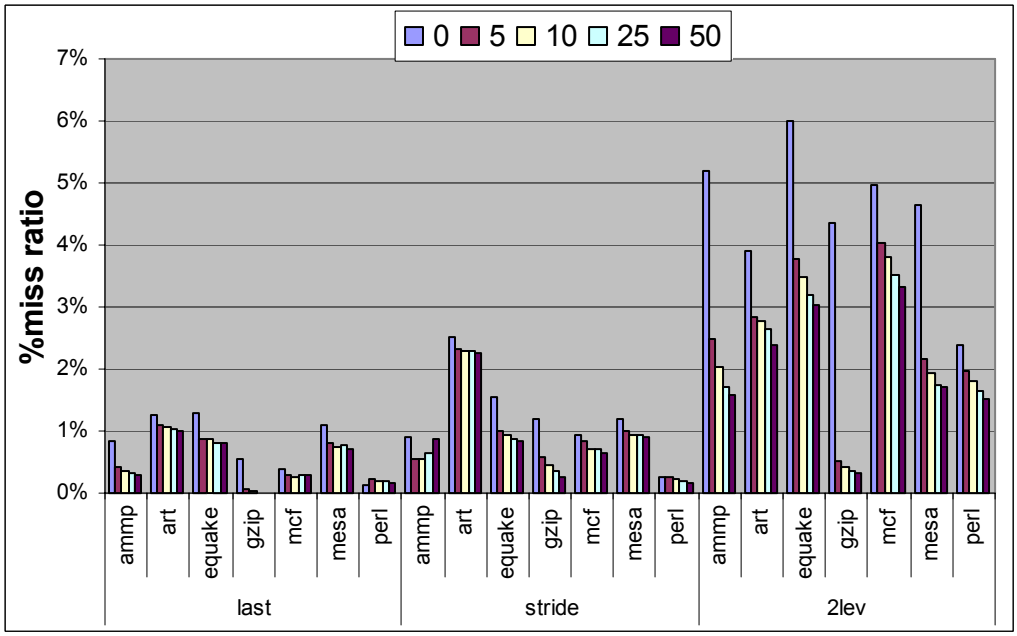

Figure 5: *MR* for other applications

4.4. The Impact of Different Table Sizes

So far we were focused on characterizing the new method using prediction tables of 1K entries. This section examines the impact of using different table sizes on these

characterizations. Due to the page limit, we will provide only the graphs for the well filtered applications, but the trend of all the other applications is the same as this group.

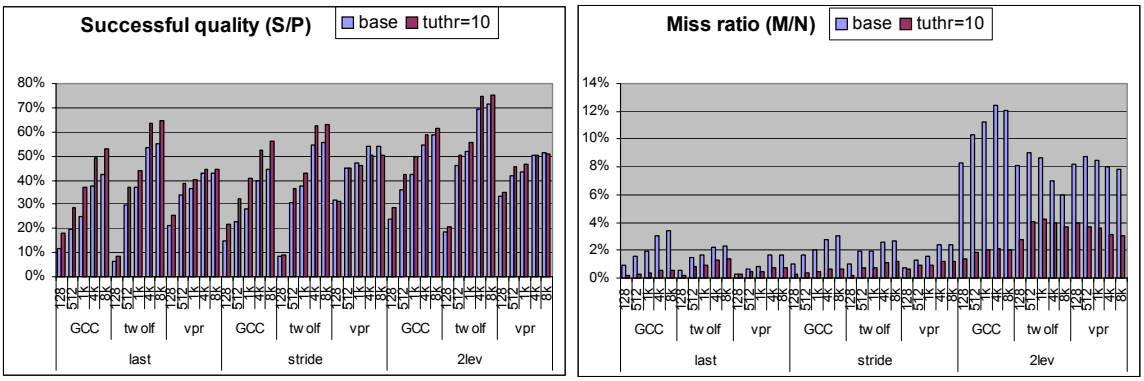

Figure 6: Impact of different table sizes

As Figure 6 indicates, when small prediction tables are used, the benefit of using value prediction is limited, but the miss rate (***MR***) is small as well. As the tables grow, the success quality(***SQ***) of the different predictors increases, with the cost of increasing the miss ratio. Luckily enough, the new proposed technique limits the growth of the miss ratio, while taking advantage of the opportunity for improving the success quality.

5. Conclusions and Future Work

This paper suggests to use hot traces as an indication for identifying "useful" candidates for value prediction, by integrating two existing mechanisms: trace cache and value prediction. The objective of this work was to answer the question "Can hot traces indicator help value prediction", or in other words, it aims to explore the new filtering mechanism, to characterize it, and to understand if an indication that a trace is hot can contribute to improving the existing value prediction mechanisms. The results presented in this paper indicate that we can define a subgroup of applications in which the use of the new technique can significantly improve the success quality of value prediction, and also to significantly reduce their miss ratio. For the applications that do not belong to this group, we observed that for most of them the current mechanisms are good enough and so no significant improvement could be achieved. But for very few applications, using the new proposed technique reduces success quality. So far, the only application that showed such "bad" behavior is GZIP which has poor success quality in the first place, but we are still working on understanding how it can be prevented and if other applications significantly suffer from that phenomenon.

Our proposed technique also has an important implication for reducing the prediction power consumption since all other techniques access the value prediction mechanism every instruction cycle (or at least for instructions of a proper type) and only if the CT table rejects their prediction, the value will not be used. In our technique, the indication that an instruction was not produced from a hot trace, will prevent it from being sent to the value prediction at all.

The work we are presenting here is the first step of an on-going research we are conducting on value prediction and trace caches. So far we have not started to examine the impact of the proposed techniques on the overall IPC of the system. In general, one can observe that any filtering technique, since it reduces the number of candidates for value prediction, may reduce the absolute number of correct value predictions, thus for a machine with infinite resources and perfect CT mechanism, there is no sense to limit the number of candidates. But, when we model "real machines", it is very important to limit the number of

speculating instructions to those that have the best chance to significantly improve the overall performance of the system, without overloading the resources with speculative instructions that have only minor impact on performance, even if predicted correctly. Although this paper does not deal with direct performance measurement issues, related research we did on characterization of hot traces, indicated that most of the instructions being executed from the hot traces do have an important contribution to the overall performance. Thus, we believe that the proposed mechanism will have an important contribution to the IPC of the machine. It will be considered as part of our future work.

6. Reference

[cald99] B. Calder, G, Reinmann and Dean Tullsen " Selective Value Prediction" ISCA 1999.

[Kos02] Oleg Kosyakovsky, Avi Mendelson and Avinoam Kolodny : The use of profile-based trace classification for improving the power and performance of trace cache systems, In workshop on feedback directed optimization, 2002.

[fiel01] B. Fields and S Rubin and R. Boodik Focusing Processor Policies via Critical-path Predictor. In ISCA 2001.

[gab97] F. Gabbay and A. Mendelson "Can Program Proofiling Support Value Prediction", IEEE MICRO 30 conference, North Carolina, 1-3 Dec, 1997

[gab98] F. Gabbay and A. Mendelson: "Using Value Prediction to increase the Power of Speculative Execution Hardware" in ACM Transactions on Computer Systems Vol. 16, No. 3 (Aug. 1998), Pages 234-270.

[lip96a] *M.H. Lipasti C.B. Wilkerson and .J.P Shen* Value locality and load value prediction in ASPLOS 96

[lip96b] *M.H. Lipasti and .J.P Shen,* Exceeding the dataflow limit via value prediction, in MICRO-29. pp 226-237 Dec 1996

[Ros01] *Rosner, R.; Mendelson, A.; Ronen, R.;* "Filtering techniques to improve trace-cache efficiency" in International Conference on Parallel Architectures and Compilation Techniques (PACT), Page(s): 37 -48, 2001

[Ros03] R. Rosner, M. Moffie, Y. Sazeides R. Ronen "Selecting Long Atomic Traces for High Coverage" to be presented in ISC03.

[rot96] E. Rotenberg, S. Bennett and J.E. Smith, "Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching", in *Proceedings of the 29th International Symposium on Microarchitecture*, Dec. 1996

[wan97] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In 30th Annual International Symposium on Microarchitecture, pages 281–290, Dec. 1997.

[tune01] E. tune D. Liang, D.M. Tullsen and B. Calder. Dynamic Prediction of critical path Instructions in HPCA 2001